Application Profile Service

Features:

Allows delegated maintenance of the UMD.

Allows delegated extension of the UMD Schema.

Information is grouped together into 'Applications'. Control of the information contained in the Application can be delegated and distributed.

Each Application contains a 'Schema'.

Information is retrieved and stored through a 'Profile' object.

Rights to the information are retrieved and stored through an 'ACL' object. This is similar to trustee rights in the directory.

Access to data is controlled by login and password.

Access control is placed on each data element in the Application. Access levels are; Read, Write, and Grant.

Each application has access controls of; SchemaAdministrator, GrantToSelf, and Scope.

Administrators can be restricted from making changes to their own object if they do not have the 'GrantToSelf' privilege.

Administrators can be 'scoped'. This is similar to the current way that administrators only have trustee rights to a subset of users in the directory.

SchemaAdministrator's are allowed to modify the Schema of the Application.

SystemLevel privileges are; AddApplication, DeleteApplication, AddExistingAttribute, AddPasswordField, SystemAdministrator.

There is a 'Public' Application. The schema includes things like the 'fullName', and email address. The Everyone Global object has been granted read rights to these fields.

Searching through the Profile or ACL is provided. This allows for finding all users who have a certain role, or set of information in their profile or ACL. Also searching by a user supplied LDAP filter is supported. The returned information is checked that you have at least read privileges.

The system supports fail-over and recovery. With dynamic clustering of AppProfile Servers. Each server supports fail-over to multiple LDAP servers. Each AppProfile Client supports fail-over to multiple AppProfile servers.

Information (Profile, or ACL) can be stored on 'Global' objects. A Global object can represent Everyone in the directory or any subset of users based on attribute matching rules.
Data is stored in a selection of AUX classes in the directory. When the schema is created the storage option is selected.

Planned AUX classes will be:
Normal – never removed.
Terminate – removed when an employee is terminated.
Move – removed when an employee terminates, or moves to another agency.

The DirXML System will be programmed to perform the AUX Class removal.

FieldTypes Supported:
Binary – Stores a single true/false setting
Selection – Stores a collection of binary selections where only one selection can be true at a time.
Option – Stores a collection ob binary selections where any number my be true at the same time.
Text – Can store any kind of text information. Can also allow multi-value storage. A new attribute is created in the UMD Schema to contain this information.
Range – Stores text information that is checked against a range restriction set on the administrator in his ACL. When granting the range field to other administrators, the Range that you give others must be equal or smaller than your own. The UMD schema is extended with new attributes for the value and range fields.
Mask – Stores text information that must conform to a mask restriction set on the administrator in his ACL. When granting the mask field to other administrators. The mask must be equal or more restrictive that your own. The UMD schema is extended to hold the value and mask fields.
ExistingText – Connects to existing attributes in the directory to give read and/or write privileges to that information.
EncryptedText – Same as normal Text except the data in the directory has been encrypted with blowfish encryption and encoded with BASE64. The prevents 'super-administrators' with direct LDAP access from seeing the data.
Password – Allows the ability to change passwords.

To add an existing Attribute or Password field to the Schema, the administrator must have that system level privilege.

The administrator who adds a field to the schema is the one who owns the rights to that field.

Code Examples:

1. Get a Connection:

Connection conn = APClient.getConnection(id, password);

Note: You should keep the connection as a static variable in your Servlet:

2. Get the account ID set by SiteMinder:

String userid = request.getRemoteUser();

3. Read Profile information:

Profile profile = conn.getProfile(userid, "MyApp");

4. Get values in Profile:

String id = profile.getString("id");
boolean administrator = profile.getBoolean("Administrator");
String[] options = profile.getStringArray("Options");
String state = profile.getString("State");

5. Update values in Profile: (Optional)

profile.setValue("Administrator", true);
profile.setValue("id", "00123");
profile.setValue("Options", new String[] {"AutoSave", "Notify", "Email"});
profile.setValue("State", "Utah");
profile.save();

Usage Scenario:

1. Tax requests an application created.
2. ITS, with addApplication rights, creates the Application
3. ITS gives all rights to the application to the administrator in TAX.
4. TAX administrator manages the schema adding any information they wish. They may also remove application rights from ITS.
5. TAX requests that the agency, home address or some other existing attribute be added to their schema.
6. ITS adds these fields into their schema. (ITS must be given the schemaAdministrator right in the application)
7. ITS gives the read or write privilege to the TAX administrator.
8. TAX does not have the 'grant' privilege so the delegation of rights to the existing field is controlled by the ITS administrator.
9. TAX administrator delegates rights to a secondary administrator by giving read and write privileges. Also the secondary administrator is given a 'scope' that limits his access to only uses in TAX.
10. The secondary administrator does not have any 'grant' privileges and cannot delegate rights further.
11. If the secondary administrator was given 'grant' to any schema elements, he could further delegate privileges to other administrators.

```
package gov.utah.das.its.appx;

import javax.servlet.*;
import javax.servlet.http.*;
import gov.utah.das.its.appx.util.*;
import gov.utah.das.its.AppProfile.*;
import gov.utah.das.its.AppProfile.Client.*;

public class Director extends HttpServlet {

   public static ServletContext sc = null;
   private static Connection conn = null;
   private static Application app = null;

   public void init(ServletConfig config) throws ServletException
   {
      super.init(config);
      sc = this.getServletContext();
      String appDN = sc.getInitParameter("UserID");
      String appPWD = sc.getInitParameter("Password");
      HttpSession session = request.getSession(true)

      try
      {
         conn = APClient.getConnection(appDN,appPWD);
         app = conn.getApplication("AppX");
      }
      catch (Exception e)
      {
            //  Some error handling
      }
   }


   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, java.io.IOException {

      HttpSession session = request.getSession(true);

      try
      {
         String userDN = request.getRemoteUser()
         Profile p = app.getProfile(userDN);

         /* Request all of the attributes contained in the profile be returned as an XML string that you can then process */
         request.setAttribute("bodyContent",XMLUtil.processXML(p.getXML()));

       --- OR ---

         /* Request individual attributes as defined in your applications schema that you setup in step 1 */
         request.setAttribute("emailAddress",p.getString("email"));
         request.setAttribute("userRole",p.getString("role"));

      }
      catch (Exception e)
      {
        //Some error handling code
      }
  }
}
```